2

Special Report
SEI-89-SR-18

Carnegie-Mellon University
Software Engineering Institute

Recommendations from the AIA/SEI Workshop on Research Advances Required for Real-Time Software Systems in the 1990s

September 13-14, 1989

William Sweet
Michael Gagliardi
Mark Klein
Reed Little
Roger Van Scoy
Robert Veltre
Charles Weinstock

December 1989

Special Report
SEI-89-SR-18
December 1989

Recommendations from the AIA/SEI Workshop on Research Advances Required for Real-Time Software Systems in the 1990s

September 13-14, 1989

William Sweet
Michael Gagliardi
Mark Klein
Reed Little
Roger Van Scoy
Robert Veltre
Charles Weinstock

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Table of Contents

List of Figures

Recommendations from the AIA/SEI Workshop on Research Advances Required for Real-Time Software Systems in the 1990s

Abstract: The Aerospace Industries Association (AIA) and the Software Engineering Institute (SEI) sponsored a workshop to facilitate clear communication between the implementors of future software-critical large systems and the professionals who sponsor or perform software-related research. The workshop was held at the SEI in Pittsburgh, Pennsylvania, on September 13-14, 1989. At the workshop, a representative group of designers of major upcoming software systems discussed among themselves and with representatives of the research community the areas of software system technology that require research advances to successfully implement these systems. The results of the discussions are summarized in this report.

1. Introduction

The Aerospace Industries Association (AIA) has identified a number of technologies critical to future space and defense systems. These technologies cover a range of disciplines, including material sciences, optics, semiconductors, propulsion systems, and advanced software. The strategy adopted by the AIA Aerospace Technical Council for the development of these technologies is based on government, industry, and academia working together to develop new concepts, increase productivity, and reduce the time to transfer technology from research to practice.

Considering the ever-increasing use of digital computers throughout all aspects of life, from commerce, to industry, to defense, it is not surprising that software technology is one of the most critical components in the AIA Key Technologies for the 1990s Program. Future challenges for software technology include building software systems with significantly increased requirements in areas having descriptions such as:

- large scale
- high performance
- distributed
- real-time
- embedded
- ultra reliable
- multi-level secure
- rapidly adaptable
- parallel

A vigorous, sustained research effort is essential to provide the skill base, tools, and methodologies needed to satisfy the software requirements of space and defense systems. Concurrently, it is necessary to direct existing or new research efforts toward the needs of the aerospace industry, and to devise a plan for effective technology transition from the research laboratories to actual industrial practice.

The effective use by industry of well-directed software research efforts is an objective of the Software Engineering Institute (SEI). The SEI joined with the AIA to sponsor a workshop to facilitate clear communication between the implementors of future software-critical large systems and the professionals who sponsor or perform software-related research. The workshop was held at the SEI in Pittsburgh, Pennsylvania, on September 13-14, 1989. At the workshop, a representative group of designers of major upcoming software systems discussed among themselves and with representatives of the research community the areas of software system technology that require research advances to successfully implement these systems. The results of the discussions are summarized in this report, which is being published by the SEI as a guide for forming and directing software research efforts in the nation.
To focus the discussion, "Parallel and Distributed Systems" was selected as the theme for the meeting because this area appears to be critical to all future software-based aerospace applications. Most workshop activity took place in small discussion groups. The workshop began with a short plenary session to set the stage for the discussions and ended with a plenary session at which the conclusions of each group were presented to all participants.

Technical representatives from AIA member companies presented concepts, requirements, and critical technology issues in future product or application areas. These presentations were conducted in parallel — one for each discussion group, six presentations altogether — as a starting point for the discussions. The presentation topics were:

1. Ultra-Reliable Underwater Vehicle
2. Fail-Safe Air Traffic Control
3. C^2 Systems Design Challenge
4. Ultra Low-Cost Brilliant Weapons Systems
5. Long-Life, Affordable Space Systems
6. Development of Integrated Avionics

The discussions of each group were not constrained to match closely the topic of the application example in the presentation, but were largely determined by each group at the time.

Participation in the workshop was by invitation only, based on lists compiled by the AIA Embedded Computer Software Committee and the SEI. The AIA committee provided the names of a representative set of persons in the aerospace industry who are engaged in the design of large complex systems. The SEI provided the names of a representative set of persons from the software research community and the government organizations that sponsor such research. In addition, each discussion group was supported by two members of the SEI technical staff, one serving as a participant and the other as report contributor. The names of the workshop attendees are listed in Appendix A.

Summaries of the six discussion groups' conclusions and recommendations are provided in Chapters 2 through 7. In Chapter 8, the issues and recommendations common to the six discussion groups are presented in summary form. Chapter 9 presents a general conclusion.

2. Ultra-Reliable Underwater Vehicle Group: Conclusions and Recommendations

This group discussed the software requirements for an ultra-reliable underwater vehicle. A typical mission scenario, a description of ocean physical characteristics, and an overview of generic mission phases were presented. The following primary vehicle requirements were discussed:

- reliability
- data storage and retrieval
- power consumption
- size and weight
- computational throughput
- command and control

The following list of initial suggested research topics was presented:

- obstacle avoidance (algorithms or heuristics)
- acoustic signal analysis
- sensor data compression and playback
- large database access mechanisms
- fault tolerant software
- software architecture
- hardware to be replaced with software
- serial sensors/actuators interconnect

Four major categories of issues were discussed:

1. education
2. the process of creating and maintaining a system (especially the software)
3. the technical nature in the domain of the discussed system
4. domain-specific technical items

The ordering of the topics in subsequent sections has no special significance. Also, all issues are reported whether or not there was a final recommended solution.

2.1. Education

This section contains issues associated with the interaction between industry and academia. They usually have differing points of view and approaches. Also, they sometimes differ on what they think the problems are. Each has its own set of valid issues; however, there is a large set of gaps and barriers.

Issues:

- The interaction between academia and industry is very spotty and sometimes contentious. There are differing points of view, approaches, and opinions on what the problems are. Both academia and industry have valid problems, but there are gaps and barriers between the two groups.
- There is no good, general-purpose mechanism for distributing technology and ideas to researchers.

- Academia trains engineers to compete with each other, but industry needs team members.

2.2. Process

This section contains issues that address the actual software development process: system design, implementation, and maintenance, and management of all three.

Issues:

- The current system design approach of hardware first will not work for the more complex systems desired in the future, i.e., a "systems" approach is needed ("software first" will not work either).
- Standard algorithm and interface building blocks do not exist or are hard to find.
- In current system building, real-time requirements are not treated as importantly as functional requirements.
- More comprehensive tools for configuration management version control are needed.
- The normal technology evolutionary path takes too long (17-19 years).
- There are several problems in both the Department of Defense (DoD) and industry with the current procurement mechanisms:
  - The wrong metrics (e.g., lines of code) are used to measure progress.
  - The current acquisition practice is geared toward the types and sizes of systems procured in the 1960s and will not scale up for the systems envisioned for the future.
  - Money instead of system development is managed.

  Usually the people organization is tied to software architecture or vice versa, which makes it difficult to effectively change one without changing the other.
- Tools that help the current approach to code generation are not enough; code generation is just part of the entire problem.
- Technology insertion (idea to application) and transfer are often difficult or impossible due to commercial competition issues.

Recommendations:

- Develop a software analog to the very large-scale integration (VLSI) research process that moves the technology into standard practice in the 10-year time frame the way the very high-speed integrated circuits (VHSIC) program does. The VHSIC program produced a great deal of good research, but the transition suffered because of the lack of an application to demonstrate the technology. Therefore, an abstraction of an application is required to provide a framework in which to present the technology. This abstraction can come in various forms:
  - a single abstraction (for research by a single group) that can then be disseminated to other researchers for further work
  - a framework for a widespread set of research — cooperative research over multi-organizations

  An example of an abstraction is multiple autonomous platforms performing cooperative tasks, which could be:

- several underwater vehicles
- several robots in a laboratory
- a group of software engineering students writing a single piece of software
- multiple subcontractors building a big widget

The framework itself should have the following attributes:

- a process-based set of models/structures for architectures or configurations of components
- instructions on how to build the framework and how to use (including evolve) it
- a set of strategies or rules describing how to use and evaluate the model
- a mechanism for modifying both the architectures and strategies
- dissemination capability

Framework selection criteria should include:

- reliability measurement
- real-time constraints measurements
- use of standard interfaces
- degree of technology insertion
- software extensibility
- amount of hardware and software required for implementation

Additional research topics related to this framework approach include:

- investigation/development of framework(s) that support the abstraction, for example:
    - layered architecture
    - International Standards Organization Open Systems Interconnection (ISO OSI)
    - real-time blackboard systems
- development of strategies and rules for using the framework, for example:
    - hardware first versus software first versus concurrent design
    - sorting out interaction of system requirements (e.g., go beyond freezing one variable at a time)
    - how to model a framework so that "what if" changes can be investigated
    - a design for software extensibility

2.3. Technology

This section presents a set of technology issues that need to be researched to successfully implement an ultra-reliable underwater vehicle. Additionally, recommendations are developed that are aimed at accomplishing the technology research objectives.

Issues:

The following areas are not sufficiently well understood to accomplish the implementation of the vehicle discussed:

- uncertainty and decision making under a problem domain
- online planning — shipboard, in route, getting back, on-vehicle mission replanning
- low-level adaptive control
- rule-based systems — deterministic, probabilistic, and non-deterministic
- coordinated vehicles (multiple control points) — both asynchronous and synchronous
- security and safety
- reliability and fault tolerance
- cost/space tradeoffs
- sensor/data fusion
- data compression/filtering versus raw bits

Recommendations:

- Emphasize horizontal research with multi-disciplined teams. This should include horizontal funding of multi-organizational, multi-disciplined teams involving government, industry, and universities, and both basic research and more applied research using a problem application and framework described previously. The applied research would stress integration of basic research topics in the application and be directed toward creating a demonstration system, not a product. This will help prove that various separate technologies interact in a supportive manner.
- Develop infrastructure mechanisms to share information locally and nationally. This could be accomplished with a national data repository that could be developed using existing infrastructures. The national data repository should be viewed as a national asset and probably be protected as such.

The following is the developed list of computer engineering research topics that are required to advance the state of the art to meet the requirements of an ultra-reliable underwater vehicle:

- fault tolerance
- vehicle online planning
- adaptive control
- data compression
- rule-based systems

2.4. Domain-Specific Research Areas

This section addresses the issues associated with the need for computer engineers knowledgeable concerning the science and engineering of underwater vehicles and who can thus be productive when employed by industry.

Issues:

- Graduate computer engineers are not knowledgeable in the basic fields of science and mathematics.
- Even fewer are educated in the field of oceanography and its related domains.
- Most cannot communicate their work in written form.

Recommendations:

- Develop a curriculum that educates computer engineers within the context of ultra-reliable underwater vehicles so that they will understand that computational systems are "a means to an end, not just an end in themselves."

- Develop a curriculum that educates computer engineers in the skills of oral and written communication.

3. Fail-Safe Air Traffic Control Group: Conclusions and Recommendations

The general problem identified by this working group is that industry cannot produce testable, reliable, affordable, and sustainable systems. The specific problems identified fall into the following four areas: process, performance and reliability, culture/people, and transition.

3.1. Process

Issues:

- The people who do the high-level requirements definition typically do not do the requirements definition for the lower level subsystems. As a result there is little or no consistency of approach, and there is no assurance that the subsystems will work well with each other. Testing becomes combinatoric, but with luck and a heroic effort spread over many months, the system might be made to work. The resulting system is nearly unmaintainable.
- The attempt to achieve consistency by putting the requirements definition under configuration control actually has the opposite effect. It becomes so painful to modify the requirements that changes are instead made to a derived requirements specification that is not under configuration control.
- Although software is at least a couple of orders of magnitude more complex than hardware the architectural discipline applied to the latter is not applied to the former. There is a misconception that software is easier to change.
- There are not enough suitable methodologies or tools that help with the design and implementation of parallel systems. Although the academic community is working on this problem, the efforts need better focus and applied research in real architectures.

Recommendations:

- When bidding a system, provide a budget for making the product "testable."

3.2. Performance and Reliability

Issues:

- There is no way to validate that a system meets a specified reliability requirement.
- Although there is a major need for a theory of reliability or dependability, academic researchers have a difficult time developing one because of the lack of data made available from industry. The academics need to be involved in the design process to collect these data.
- System reliability is hard to test because of the difficulty of injecting errors realistically. Building tests into the delivered system may actually make the system less reliable because of increased complexity.
- Systems are not instrumented to provide information about their performance and reliability as they operate. This is somewhat like trying to fly an airplane with no instruments. These runtime metrics should be a part of the system requirements and not "overhead," but they are usually not a deliverable under the contract.

- Even if systems were instrumented, metrics of performance and reliability need to be developed. Where the performance or reliability of a component of a system can be measured, it seldom says anything about the performance or reliability of the system as a whole.
- Since there are no good metrics, there is no way to determine the tradeoffs between performance and reliability, and other factors such as time to develop, final cost, weight, power consumption, etc.

Recommendations:

- Design methodologies and tools to support "design for testability" for software reliability and performance. This would be at both the requirements analysis level and the system design level.
- Bootstrap current visualization techniques for software instrumentation. Use this and other instrumentation to develop models of reliability and performance prediction and validation. Instrumentation must be left in the delivered product to facilitate reliability and performance measurement.
- Using metrics developed above, provide a means of determining the tradeoffs between performance and reliability and other quantifiable characteristics of a system such as memory usage and power consumption.

3.3. Culture/People

Issues:

- Software designers are not involved in the design process early enough. The domain experts view software designers as members of a "priesthood." The software designers attempt to interpret the experts view of the field with various degrees of success.
- Academic research usually has a unidimensional thrust. It is easier to get results by holding every dimension but one of an experiment constant. The bean-counting approach to academic promotion contributes to this. Real-world problems are often much more multidimensional.
- The people who begin a major aerospace project are seldom the ones who finish it. There is a high turnover and a loss of continuity. This is due to both project duration and to the periodic assignment of "specialists."

Recommendations:

- Set up reward structures that encourage industry to "pull" in new technology and the academic community to "push" their results. An in-house individual should be responsible for pulling technology into the company.
- Encourage academic researchers to publish their results in applicable trade journals as well as professional journals.
- Reward academic researchers for multidimensional research and discourage unidimensional research.

3.4. Transition

Issues:

- Aerospace developers do not have time to keep up with academic developments. They are so driven by schedules that they are lucky to keep up with trade journals. This makes it difficult for them to learn about developments that might be useful to their system development efforts.
- There are major barriers to reuse of software:
    - Technical - "design for reusability" and retrieval techniques are immature.
    - Cultural - the so called not-invented-here attitude.
    - Liability - if a lawsuit is brought involving a system with reusable components, who is legally exposed?
    - Legal - it is difficult to share software among different contracts.
    - Greed - the opportunity to make more money by reinventing the wheel.

  Because of tight specifications such as maximum memory or power consumption, there is a tendency not to reuse a piece of software that does more than is needed because of the possible waste of resources.
- Universities are not good at disseminating their technology to industry. The model of a professor starting a company to do the transition has two difficulties: 1) it effectively takes the researcher out of the research business; and 2) such companies have a dismal history of success.
- No individual involved with a project is held accountable for implementing advanced technical changes. There are no incentives to keep up with the latest research results, much less to use them. On the contrary, this introduces risk, which is anathema to aerospace engineers and managers.

Recommendations:

- Establish working partnerships between industry and academia early enough to capitalize on new and current research:
    - Use the Software Engineering Institute (SEI) and the Aerospace Industries Association (AIA) as a catalyst for establishing partnerships.
    - Tie the partnership to a specific technical domain rather than to a specific system or project.
    - Involve multiple participants in the partnerships, not just a single industry-academic pair.
- Identify and use, as a foundation, research that already exists. For instance, capitalize on the emerging theory of scheduling to derive subsystem-level design.
- Use the SEI to aid in identifying and disseminating appropriate technology. Ensure that there is a feedback loop between industry and academia.
- Develop standards and a demonstrable prototype capability that may be incorporated in products.

4. C^2 Systems Design Challenge Group: Conclusions and Recommendations

Command and control (C^2) is defined as the exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of the mission. Typical large C^2 systems are widespread and deep with respect to their functions and connectivity, spanning a range of services and can be international is scope. The systems are heavily software and data intensive, real-time, secure with massive amounts of data principally received from sensors and communication elements. A major C^2 challenge is developing affordable systems while meeting ever increasing requirements. These requirements often conflict, complicating the design and degrading the performance. An example of this dilemma is the conflict between an open system and the security needed to protect the system. The challenge is to find the right balance that meets system requirements at minimum cost.

4.1. Conceptualizing Large Complex Systems

Issues:

- There are no notations available for adequately describing the large complex systems of the 1990s. In addition, there are no techniques that allow the system designer to think about and manipulate the system as an abstract object, i.e., there are no early life-cycle thought tools.
- There is no approach for software tradeoff analysis. There are no criteria for partitioning a system between hardware and software (as in trading software for special purpose hardware).

Recommendations:

- Develop a new level of abstraction, one that can encompass and characterize a system (the properties of systems, how they interact, and how they are manipulated).
- Develop a theory for dealing with the decomposition and recomposition of systems (a theory of composability). Develop techniques and tools to study system integration.
- Develop a means for studying the different system aspects in an integrated manner.
- Develop approaches for insuring the application of concepts across a large system in a consistent manner.
- Develop techniques and tools to model subsystems and systems.
- Develop techniques for performing software tradeoff analysis across functionality, performance, openness, and trustworthiness.

4.2. Requirements Generation

Issues:

- Requirements in specifications get detailed too quickly, lacking sufficient presentation of the operational requirements of the system. Designers need a good understanding of the intended operational usage of the system.

- Current functional approaches to requirements generation result in fragmentation. There are no good techniques for achieving traceability of requirements, clean partitioning of requirements, and determining the feasibility of requirements. Plus, functional requirements do not map well into object oriented design approaches.
- Time and performance requirements are not adequately expressed and even when expressed, are not readily testable, and most certainly cannot be verified.

Recommendations:

- Develop formal languages for specifying requirements and techniques for verifying requirements. This needs to go beyond simply verifying that the formal requirements are consistent among themselves. It must begin to address the correctness of the requirements in stronger terms: do the requirements solve the user's problem?
- Develop techniques and tools for generating requirements. In particular, approaches for the specification of time and performance requirements are critical. Another critical aspect of this problem are techniques that help in the the distribution of timing requirements over a distributed system.
- Develop techniques to measure, test, and verify performance.
- Investigate the applicability of object-oriented design approaches to distributed real-time systems.
- Develop techniques to incorporate time into object-oriented requirements and design.

4.3. Software Management Process

Issues:

- The software management process is based on the wrong model. Software is developed the way hardware used to be. It is designed, built, tested, and maintained. Unfortunately, due to the pliable nature of software, design is never finished, building goes on as long as the software is used, testing never finds all the problems, and maintenance starts with the the first line of code.
- The software industry is using 1960s technology to develop systems for the 1990s and beyond.
- Software engineers are undercapitalized (tools, methods, etc.).
- There is insufficient automated support for the software management process.

Recommendations:

- Identify and test new models for the software development process.
- Evaluate the impact of prototyping on the software life cycle.
- Consolidate existing software research into a usable form and transition this information to industry.
- Provide incentives to the software industry to provide the tools needed for more productive software engineers.
- Define a standard set of software metrics. Then use these metrics to develop a set of management tools.
- Integrate these management tools into software development environments.
- Educate people about the *real cost* of making changes in delivered software.

4.4. Integrating Disparate Attributes

Issues:

- Large systems increasingly require interactive attributes that frequently conflict with each other (e.g., security and openness). There needs to be a means to integrate the attributes of these subsystems smoothly across the entire system. Some sample attributes include:

    - performance (real-time, throughput, response time, etc.)
    - reliability (availability)
    - security (multi-level security)
    - openness (standards)

  These attributes can interact in subtle and unexpected ways. The solution to this problem must involve development processes, software tools, and methods. One additional difficulty with this problem is the traditional research view of looking vertically (deep) into one particular specialty. whereas, a more horizontal view of research is required to look across all the required system attributes.

Recommendations:

- Step 1 (within one domain) => a case study cookbook for the domain

    - Collect an experience database, collect historical records of project development, survey existing approaches
    - Perform domain analysis
    - Log system engineer's lessons learned
    - Establish a vocabulary
    - Understand the simulations of the domain

- Step 2 (still within one domain) => moving from an art to a science

    - Analyze and synthesize the experience
    - Develop a theory and models to express the domain
    - Develop tools for manipulating the theory and models
    - Use advanced simulation techniques for studying the system

- Step 3 => begin to generalize across multiple domains to evolve a true theory of systems

4.5. Automated Development Environment

Issues:

- The development of the large complex systems of tomorrow is very difficult given the current way of doing business. There is a general lack of automated software generation aids in industry today. Every line is written anew. What is needed is a revolutionary software development environment that integrates the entire software life cycle. Such an environment might look like the one in Figure 4-1:

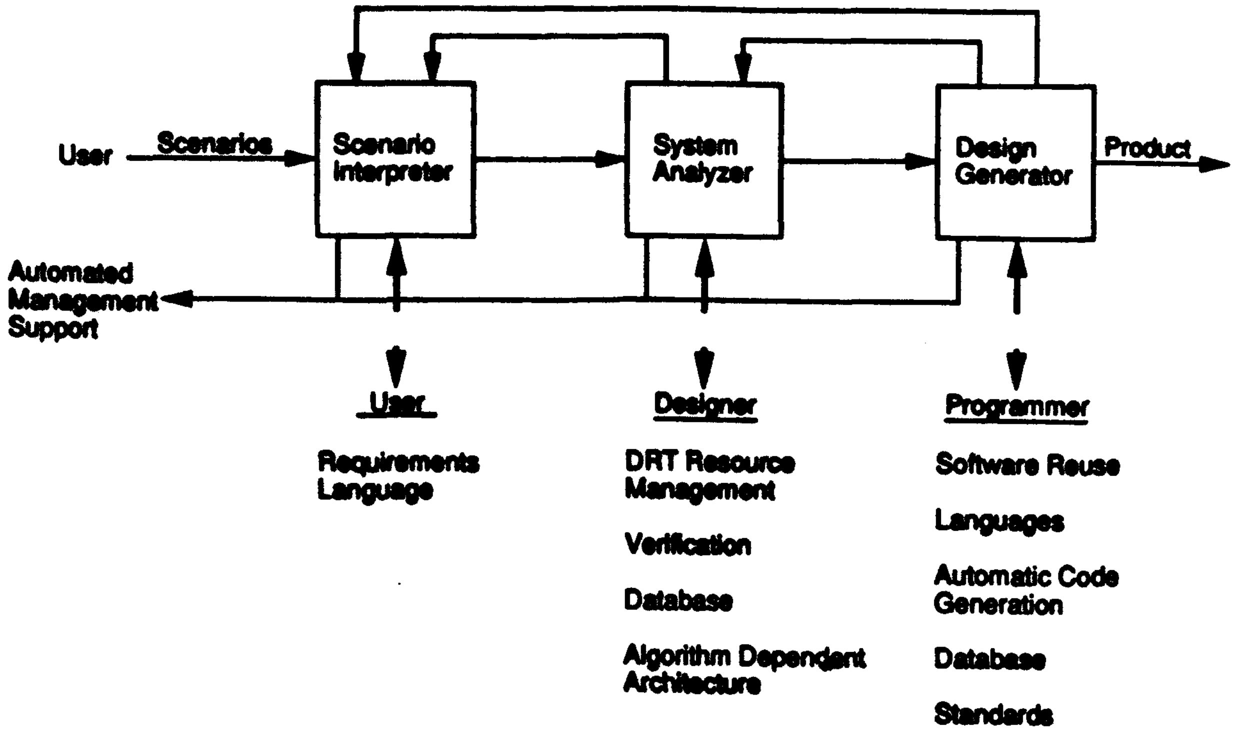


Figure 4-1: Automated Development Environment

Recommendations:

- Define a framework for merging different modules shown above.
- Develop individual components of the integrated environment shown above.
- Define a framework for extending the environment shown above.

4.6. Software Reuse

Issues:

- It is difficult to separate the discipline and management problems of reuse from the technology and research problems.
- There are no standards governing the interfaces across reusable components.
- There is a significant overhead to creating reusable software. Adding the obscurities of software data rights creates a business environment that is not conducive to the creation and application of reusable software.

Recommendations:

- Collect feedback on actual reuse experiences to identify the real problem areas.
- Develop techniques for identifying potentially reusable parts.
- Develop a framework for integrating components. This framework can then be evolved into a standard.
- Clarify the software data rights problems. Provide incentives to industry to move toward increased reuse.

4.7. Testing and Verification

Issues:

- The functional testing of large, distributed, real-time systems is extremely difficult. The addition of timing and performance requirements compounds the problem. Testing is currently used to "verify" systems, not by preference, but because the only alternative, formal verification, is not possible for real systems. What is needed is the ability to verify designs, not code. The leverage comes from eliminating errors before they become code, rather than verifying that the code is incorrect.

Recommendations:

- Develop techniques that allow designs to be verified. This reduces to determining how formality can be applied earlier in the life cycle (i.e., much earlier than the code statement level) and encompasses the development of formal design and specification languages.
- Develop techniques and tools that make verification viable for large real systems.
- Develop a theory for testing distributed systems.
- Develop techniques that capture the "ilities" (reliability, enhanceability, security, etc.) in a way that can be tested.

4.8. Database Technology

Issues:

- There is a lack of effective data consistency, concurrency, and synchronization for distributed, real-time database systems.
- There is poor incorporation of time-based data into the database.
- The performance of existing database technology is not good enough for the systems of tomorrow.

Recommendations:

- Develop temporal databases.
- Develop object-oriented databases.
- Develop a theory of approximate query response.
- Develop techniques that allow the integration of semantic information into database systems.

4.9. Distributed Real-Time Resource Management

Issues:

- There is no distributed real-time operating system over heterogeneous machines.
- The currently mandated language does not support distributed code.
- No techniques exist for the specification and verification of real-time properties of systems.

Recommendations:

- Develop a theoretical basis for multi-resource scheduling.
- Develop techniques and principles to support construction of distributed, real-time systems.

4.10. COTS Interoperability

Issues:

- Incompatibility between commercial off-the-shelf software (COTS) forces much of the existing computational resources to be consumed in simply making diverse machines and software talk to one another.

Recommendations:

- Define a binary standard to achieve bit-level data compatibility (data interchange standard).
- Define a standard interface between computers.

4.11. Defects in the Acquisition Process

Issues:

- There are defects in the research and development (R&D) funding process. There is a definite lack of long term direction in R&D funding. What should the funding priorities be? Who should set the funding priorities? Who should get funding (university versus industry versus MITRE-style organizations)?
- One specific defect stems from the reluctance of acquisition agents to "sign-off" on any software-related item until the code is available and tested.
- How can the research community be encouraged to address the problems that are most urgent for industry?

Recommendations:

- Stabilize long-term funding for basic research.
- Develop techniques and tools that will give acquisition agents the confidence that early life-cycle products are acceptable and correct; educate acquisition agents to accept these verification techniques.
- Educate acquisition agents about the advantages of evolutionary development and delivery (i.e., that every system is "incomplete" at delivery and is continually changing over time).
- Develop a cooperative interface between government, industry, and academia, one that allows academia to receive input from industry and industry to more readily access and apply the results from academia.

5. Ultra Low-Cost Brilliant Weapons Systems Group: Conclusions and Recommendations

This group organized discussions around the following key areas:

- Paradigms - How are the problem space and solution space viewed?
- Life Cycle - What are the steps to solving the problem?
- Tools - How can the steps in software development, use, and maintenance be automated?
- Databases - What database technologies can be applied to the development of operational systems?
- Industry/Funding Agency Objectives - What is the role of the industry/funding agency?

The following sections contain the issues and recommendations for each of these key areas.

5.1. Paradigms

Issues:

- Current paradigms do not adequately support real-time distributed and parallel processing system needs. For example, existing design paradigms such as functional decomposition, data flow, data structure, and object orientation are inadequate in representing large-scale real-time systems.
- New ways of looking at large real-time distributed systems design problems are needed that require "lateral thinking," better abstractions, separation of concern, and management of complexity.
- A standardization of paradigms needs to be developed, disseminated, and put into practice.

Recommendations:

- Elucidate the problems using paradigms.
- Demonstrate the usefulness of the paradigms by showing that they scale up to large systems, map onto real problems, and can become automated tools to enhance productivity.
- Reduce total application risk by the rigorous application of paradigms.
- Provide a metrics to compare the utility of alternate paradigms.

5.2. Life Cycle

Issues:

- Software is an integral component of a system and must be treated on the same level and with the same importance as hardware throughout the life cycle of the system.
- Software requirements are often unknown in the early stages of system development and are not usually specified in the system requirements. This leads to many unanswered questions. Is there a sufficient requirements model? How do system requirements map to software requirements? How are software partitioning decisions derived?

- The current life-cycle paradigms may not adequately address the needs of large-scale real-time systems. What is the appropriate life-cycle paradigm? Are software life-cycle paradigms different from system or hardware life-cycle paradigms? If so, how are they different?

Recommendations:

- Investigate what is needed in the area of software archeology, i.e., what is successful, what is not, and why. There is a wealth of data pertaining to this but the data are often not comparable. When they are, they have not been correlated to extract useful information.
- Develop better ways of estimating software costs and schedules for large-scale systems.
- Develop requirements traceability methods throughout the life cycle of the system.

5.3. Tools

Issues:

- Tools should support automation of chosen design methodologies and practices; they should not dictate which methodology or practice to employ.
- Tools should be integrated into the system at all levels of the system life cycle and be capable of supporting large-scale systems.
- Tools should be used for processes that are well understood.
- The current state of practice requires heavy capital investment and is not market or demand driven. Computer-aided software engineering (CASE) tools have been oversold and are not at a mature level.

Recommendations:

- Standardize methodologies to make tools practical and useful in large real-time systems.
- Develop paradigms for program generation tools used in code development and testing.
- Develop tools to capture past systems design experiences and perform forward and reverse engineering.
- Develop evaluation criteria to quantify the benefits of tools to assess the cost and benefits of the tools.

5.4. Databases

Issues:

- Software development necessitates traceability of data objects throughout the life cycle of software. This implies the need for a data object descriptions database.
- Configuration management techniques are needed to handle large software systems that may be highly distributed. Also, security levels and protection mechanisms must exist.
- Operational software systems need to address the problems of distributed databases. The question of how to best distribute the data must be answered, for example, file server model versus distributed data model. Other areas of concern are data replication, consistency, and senescence.

Recommendations:

- Provide object databases that maintain the integrity of the data object and software interfaces during software development.
- Orient the database research to address real-time performance issues as well as operational data integrity and fault tolerance.

5.5. Industry/Funding Agency Objectives

Issues:

- The industry/funding agency should help define and agree on case studies and test cases to present to researchers and also should better classify the problem domains.

Recommendations:

- Maximize the competitive spirit among researchers and create a high return on investment for research money. The apparent success of the Gordon Bell awards for parallel algorithms is an excellent example.

6. Long-Life, Affordable Space Systems Group: Conclusions and Recommendations

This working group identified key problems associated with building space systems, particularly in areas in which successful research would have to be performed to ensure the timely and economical development of future space systems. A set of criteria for evaluating the utility of performing research in the selected areas was developed, and each research area was evaluated against these criteria.

The following sections present the major issues identified and the recommendations offered by the working group, a set of criteria used to evaluate proposed research areas, and the results of the working group's evaluation.

6.1. Predicting and Validating System Correctness

Issues:

- In nearly all cases, once a space system is deployed there is no mechanism for retrieving the system to make corrections to the operational system. Therefore, it is imperative that the correctness of the space system be known prior to its deployment.

Recommendations:

- Perform research in the areas of *predictability and validation* with the goal of developing methods and tools in support of a systematic approach to predicting the correctness of a space system prior to the system's deployment. Such methods and tools would be used to validate the functional and performance correctness of the system and should be applicable to all phases of system development, from requirements specification to certification.

6.2. Autonomous Operations

Issues:

- A deployed space system must be able to survive and continue operation in the presence of both hardware and software failures. The current trend is toward an "operate through" philosophy, which requires increasingly sophisticated and autonomous on-board failure detection and correction to allow the system to continue performing its mission at all times regardless of failures.

Recommendations:

- Perform research in the area of *autonomous systems* with the goal of defining software and hardware architectures that permit increasing levels of functional fault tolerance through software and hardware fault tolerance.

6.3. Evolutionary Systems

Issues:

- Space systems evolve over time and continue in operational use for 10 to 20 years or longer. During the operational period, elements (including new spacecraft or ground processing stations) are added to replace failed elements or to increase functionality. New and old elements must interoperate. Methods and tools for assessing and minimizing the impact of proposed changes to systems are ad hoc or altogether lacking.
- Because space systems evolve, the original developers of a system are typically unavailable to perform system upgrades or to correct problems. Methods and tools for capturing the original developer's rationale for the design and implementation of system components are ad hoc or lacking.
- Space systems tend to be tightly bound to the specific technologies used to implement them. Thus, it is typically a very difficult and expensive process to insert new technologies into evolving systems.

Recommendations:

- Perform research in the area of *evolutionary systems* with the goals of defining software and hardware architectures conducive to an evolutionary approach to system development.
- Prescribe a life-cycle model, methods, and tools to support the evolutionary development of systems.
- Prescribe methods for evolving systems transparently and on the fly.
- Define a requirements specification system that would permit a quantitative assessment of the estimated impact a change request would have on the areas most difficult to assess, such as system reliability, revalidation, and cost to incorporate.

6.4. Reusable Components

Issues:

- Reusing previously validated and certified components of space systems would go a long way toward mitigating the cost and difficulty associated with validating the correctness of a space system prior to its deployment. Yet there are a number of issues related to reuse that must be resolved before reuse can be incorporated into the process of developing complex space systems.

Recommendations:

- Perform research in the area of *reuse* with the goals of defining what types of objects can be reused (e.g., requirements specifications, certified flight code, validation test cases).
- Define techniques for packaging reusable components (e.g., libraries).
- Define schemes for acquiring knowledge of and locating reusable components.
- Prescribe methods and tools for designing and developing systems with eventual reuse in mind (e.g., domain analysis, specification and design methods).
- Define methods and tools for capturing the rationale associated with the design and implementation of system components.

6.5. System Connectivity

Issues:

- As space systems employ distributed processor architectures and themselves form part of even larger space networks, the issues associated with distributed processing and distributed databases present even greater challenges to the process of determining the correctness of a system prior to its deployment.

Recommendations:

- Perform research in the area of *connectivity* with the goals of defining methods and tools for transparently distributing applications across distributed processors.
- Define strategies that yield optimal distributions of applications across processors.
- Define strategies and techniques for performing dynamic reallocation of computing resources amongst the consumers of the resources.
- Define strategies and techniques for ensuring that security can be maintained throughout a network.
- Define strategies and techniques for managing the use of redundant computing resources in the event of failures.

6.6. Simulation and Test

Issues:

- Simulation and test are currently the predominant methods for determining the correctness of a space system before the system is deployed. But as the complexity of actual space systems increases, the ability to determine correctness through simulation and test becomes progressively more costly and difficult to achieve.
- At each phase in the system development life cycle, it is becoming increasingly more costly and difficult to perform verification and validation.
- At each phase in the system development life cycle, it is becoming increasingly more costly to correct flaws that were not uncovered in earlier phases.

Recommendations:

- Perform research in the areas of *simulation and test* with the goals of defining a software simulation and test environment that is accurate to the level of the target instruction set architecture yet flexible enough to support testing.
- Define standard simulation systems for complex distributed target architectures and define standard test management systems.

6.7. Effective Use of Research Resources

Issues:

- The period of time for a major technological change to move from concept to operational use in a space system is currently anywhere from 10 to 20 years.
- The organizations responsible for building space systems are not always aware of applicable research being performed at universities or consortia. Likewise, researchers are not always aware of the organizations that could potentially benefit from their research.

Recommendations:

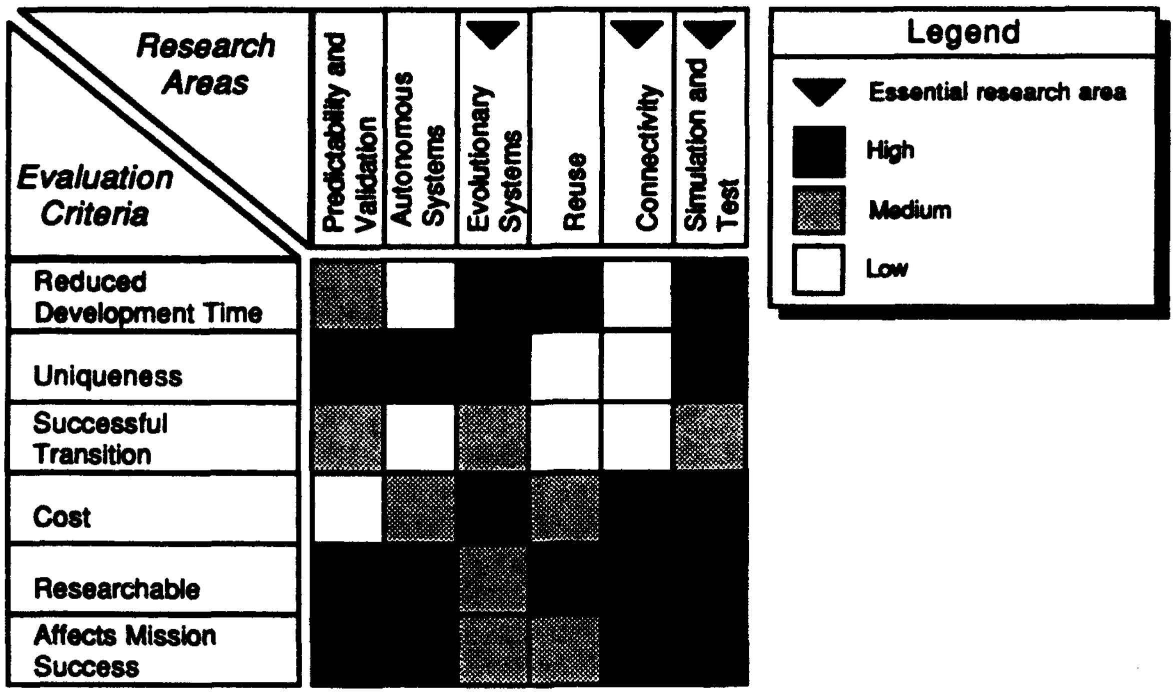
- Define mechanisms for matching suppliers of research and technology with potential users of such research and technology at both the national and corporate levels.
- Define mechanisms to widely disseminate positive applications of advanced technologies as well as the lessons learned from attempting to do so.
- Define mechanisms for rapidly disseminating knowledge of technological and research advances.

6.8. Research Area Evaluations

The working group decided upon the following criteria to evaluate the utility of the six research areas proposed in the previous sections:

- The probability that applying successful research in this area to the development of future space systems would yield shorter system development times than would have been possible without the research advances (*reduced development time*). A research area evaluated against this criteria is assigned a value from *high*, *medium*, and *low* (high probability, medium probability, low probability).
- The degree to which research in this area is unique, that is, a measure of the extent to which research in this area is being performed (*uniqueness*). A research area evaluated against this criteria is assigned a value from *high*, *medium*, and *low* (highly unique research area, etc.).
- The probability that research in this area will be successfully applied to the development of a real space system (*successful transition*). A research area evaluated against this criteria is assigned a value from *high*, *medium*, and *low* (high probability, medium probability, low probability).
- An estimation of the magnitude of the cost required to perform successful research in this area (*cost*). A research area evaluated against this criteria is assigned a value from *high*, *medium*, and *low* (high cost, medium cost, low cost).
- The extent to which meaningful research can be performed in this area. This takes into account both the ability to do meaningful research and the willingness of someone to do the research (*researchable*). A research area evaluated against this criteria is assigned a value from *high*, *medium*, and *low* (highly researchable, etc.).
- The extent to which successful research in this area will have a positive effect on mission success (*affects mission success*). A research area evaluated against this criteria is assigned a value from *high*, *medium*, and *low* (highly positive effect, etc.).

The results of the evaluation are shown in Figure 6-1. After performing the evaluation, the working group concluded that all of the research areas were useful, some more than others. As shown in Figure 6-1, the working group also determined that research in the areas of *evolutionary systems*, *connectivity*, and *simulation and test* is essential to the future successful development of complex space systems.



Evaluation Criteria / Research Areas	Predictability and Validation	Autonomous Systems	▼ Evolutionary Systems	Reuse	▼ Connectivity	▼ Simulation and Test
Reduced Development Time	▨	□	■	■	□	■
Uniqueness	■	■	■	□	□	■
Successful Transition	▨	□	▨	□	□	▨
Cost	□	▨	■	▨	■	■
Researchable	■	■	▨	■	■	■
Affects Mission Success	■	■	▨	▨	■	■

Legend	
▼	Essential research area
■	High
▨	Medium
□	Low

Figure 6-1: Evaluation of Selected Research Areas

7. Development of Integrated Avionics: Conclusions and Recommendations

This working group discussed issues related to the development of integrated avionics, and divided the issues and recommendations for improvement into five main categories: timing, maintenance, reliability, security and schedule.

7.1. Timing

As complexity increases, it becomes extremely important to be able to clearly and unambiguously specify timing requirements. Additionally, real-time systems must be designed by considering timing requirements and characteristics at the outset.

Issues:

- *Methods* are needed to formalize the management of time in real-time systems. Methods either do not exist or are not widely known. Formal specification of real-time requirements are needed. Analytical methods are needed to understand the timing behavior of existing applications and to understand and predict the timing behavior of applications that are being developed.
- *Tools* are needed to facilitate the practical application of analytical methods. It is important to be able to raise timing issues as early as possible in the development cycle. Executable specifications would allow for simulations at a very early stage of the life cycle. Tools based upon analytical models would enable early analysis and would allow designers to ask "what-if" questions concerning the timing aspects of the system.
- *Instrumentation techniques* are needed to empirically verify theoretical predictions. Instrumentation must either be non-intrusive or predictably intrusive. Instrumentation must be considered during design, not as an afterthought during testing.

Recommendations:

- Transition the rate monotonic scheduling theory, a well-established and growing body of theory that provides an analytical framework for managing time in real-time systems. This theory is being developed at the Software Engineering Institute (SEI) and Carnegie Mellon University (CMU).
    - Continue to extend the theory to distributed systems.
    - Continue to demonstrate the practical applicability of the theory to actual systems.
    - Develop tools to facilitate its applicability.

7.2. Maintenance

Software systems evolve during development and after deployment. New requirements, a better understanding of old requirements, and advances in hardware and software technology contribute to the need to evolve. This system evolution mandates the need for tools and methods to understand existing systems and to verify the integrity of the modified system.

Issues:

- ***Real-time software engineering environments*** that cater to the incremental development and maintenance of distributed real-time embedded systems are not available. The environment that is used by the original developers needs to be available to the maintainer. The environment must support a formal representation of requirements and a representation of the software artifact that allows it to be analyzed and viewed from various perspectives. This should include timing analysis tools, test generation tools, and the ability to view the artifact at various levels of abstraction textually and graphically. The environment must support the rapid generation of associated documentation and traceability through all phases of the life cycle.
- *Verification techniques* are needed to guarantee the integrity of the modified system. Software engineering techniques and methods, such as modularity and information hiding, need to be employed to increase the modifiability of systems by localizing the effects of change, thus preventing the propagation of an error that may have been introduced by an upgrade. Techniques are also needed to verify the timing correctness in addition to the functional correctness of a modified system when new hardware and/or software technology is introduced. Techniques are needed to ensure that tools do not introduce errors into the the system.

Recommendations:

- Educate practitioners in basic software engineering methods.
- Establish the theoretical foundations that will allow for the construction of a comprehensive software engineering environment that supports the development and maintenance of distributed real-time systems.

7.3. Reliability

Reliability of life-critical systems is critical to measure. When systems fail, the failure must be detected, logged, and tolerated.

Issues:

- Adequate theory and techniques do not exist for performing quantitative reliability assessment.
- Adequate theory and techniques do not exist for the design of reliable systems. General proofs of correctness are not yet practical to ensure correctness. Extremely comprehensive testing is necessary to compensate for the lack of analytical techniques. Techniques are needed to gather data about faults to determine the circumstances and conditions that lead to a particular fault.

Recommendations:

- Extend and develop theory that allows for the accurate assessment of mean time between failures.
- Place an emphasis on establishing classes of faults where specialized knowledge can be employed to increase reliability and detect and circumvent faults (e.g., in the area of timing faults, scheduling theory may offer analytically based strategies for fault detection and correction).

7.4. Security

Security is an issue in both the commercial and military arenas. Protection against intentional harm and the security of classified data are concerns.

Issues:

- Procedures, policies, and techniques are not adequate to prevent computer-related crime. Techniques for detecting time bombs, viruses, worms, and the like are needed. Software development and maintenance procedures have to be developed with potential sabotage in mind when life-critical systems are concerned.
- With a trend toward integrated avionics, the separation of classified and unclassified data becomes an important issue.

Recommendations:

- Establish a base of theory that leads to a set of criteria for the design and certification of secure systems.

7.5. Scheduling

The development schedule for real-time systems tends to be difficult to estimate and is frequently too long.

Issues:

- Better metrics, cost-estimating techniques that are based upon those metrics, and tools to support the techniques are needed to facilitate the project management aspects of developing real-time software.
- Real-time software project management must be able to accommodate rapid prototyping and incremental software development.

Recommendations:

- Extend the theoretical foundation for project management.
- Encourage the creation of associated project management tools to support incremental development of real-time systems.

8. Summary of Research Recommendations

A number of common themes spanned the six group discussions. These common areas are:

- system and software reliability
- requirements specification, testing, and verification
- software engineering life cycle, processes, and methodologies
- software reuse
- distributed databases
- fault tolerance
- software engineering education
- technology transition

For each common area, we present a brief summary of the essential issues and reiterate the main recommendations presented in the previous six chapters.

8.1. System and Software Reliability

Several of the discussion groups identified issues and recommendations related to system and software reliability. The recommendations in this area addressed issues such as the lack of a general theory of reliability or dependability, the difficulty of expressing reliability requirements and verifying systems against those requirements, and the lack of adequate techniques for performing quantitative reliability assessment. The key recommendations, presented in previous sections, are reiterated below:

- Construct methodologies and tools to support "design for testability" for software reliability and performance. This would be at both the requirements analysis level and the system design level.
- Bootstrap current visualization techniques for software instrumentation. Use this and other instrumentation to develop models of reliability and performance prediction and validation. Instrumentation must be left in the delivered product to facilitate reliability and performance measurement.
- Using metrics developed above, provide a means of determining the tradeoffs between performance and reliability and other quantifiable characteristics of a system such as memory usage and power consumption.
- Extend and develop theory that allows for the accurate assessment of mean time between failures.

8.2. Requirements Specification, Testing, and Verification

Several of the discussion groups identified issues and recommendations specific to the interrelated areas of requirements specification, testing, and verification. The recommendations in these areas addressed a broad range of issues, including deficiencies with functional approaches to requirements generation, partitioning of requirements, traceability, the inadequacy of exhaustive testing for distributed real-time systems, and the cost and complexity associated with testing and verifying distributed systems. The key recommendations, presented in previous sections, are reiterated below:

- Develop formal languages for specifying requirements and techniques for verifying requirements. This needs to go beyond simply verifying that the formal requirements are consistent among themselves. It must begin to address the correctness of the requirements in stronger terms: do the requirements solve the user's problem?
- Develop techniques and tools for generating requirements. In particular, approaches for the specification of time and performance requirements are critical. Another critical aspect of this problem are techniques that help in the the distribution of timing requirements over a distributed system.
- Develop techniques to measure, test, and verify performance.
- Develop techniques to incorporate time into object-oriented requirements and design.
- Develop techniques that allow designs to be verified. This reduces to determining how formality can be applied earlier in the life cycle (i.e., much earlier than the code statement level) and encompasses the development of formal design and specification languages.
- Develop techniques and tools that make verification viable for large real systems.
- Develop a theory for testing distributed systems.
- Develop techniques that capture the "ilities" (reliability, enhanceability, security, etc.) in a way that can be tested.
- Develop requirements traceability methods throughout the life cycle of the system.
- Perform research in the areas of *predictability and validation* with the goal of developing methods and tools in support of a systematic approach to predicting the correctness of a space system prior to the system's deployment. Such methods and tools would be used to validate the functional and performance correctness of the system and should be applicable to all phases of system development, from requirements specification to certification.
- Define a requirements specification system that would permit a quantitative assessment of the estimated impact a change request would have on the areas most difficult to assess, such as system reliability, revalidation, and cost to incorporate.
- Perform research in the areas of *simulation and test* with the goals of defining a software simulation and test environment that is accurate to the level of the target instruction set architecture yet flexible enough to support testing.
- Define standard simulation systems for complex distributed target architectures and define standard test management systems.

8.3. Software Engineering Life Cycle, Processes, and Methodologies

Almost all of the discussion groups identified issues and recommendations related to the software life cycle and software development processes and methodologies. The recommendations in these areas addressed a broad range of issues, including the lack of suitable methodologies for designing distributed real-time systems, insufficient automated support for the software management process, and the need for alternative life-cycle models that support an evolutionary approach to system development. The key recommendations, presented in previous sections, are reiterated below:

- Develop a theory for dealing with the decomposition and recomposition of systems (a theory of composability). Develop techniques and tools to study system integration.
- Develop techniques for performing software tradeoff analysis across functionality, performance, openness, and trustworthiness.

- Investigate the applicability of object-oriented design approaches to distributed real-time systems.
- Identify and test new models for the software development process.
- Evaluate the impact of prototyping on the software life cycle.
- Define a standard set of software metrics. Then use these metrics to develop a set of management tools.
- Integrate these management tools into software development environments.
- Develop techniques and principles to support construction of distributed, real-time systems.
- Investigate what is needed in the area of software archeology, i.e., what is successful, what is not, and why. There is a wealth of data pertaining to this but the data are often not comparable. When they are, they have not been correlated to extract useful information.
- Develop better ways of estimating software costs and schedules for large-scale systems.
- Develop evaluation criteria to quantify the benefits of tools to assess the cost and benefits of the tools.
- Prescribe a life-cycle model, methods, and tools to support the evolutionary development of systems.
- Perform research in the area of *connectivity* with the goals of defining methods and tools for transparently distributing applications across distributed processors.
- Establish the theoretical foundations that will allow for the construction of a comprehensive software engineering environment that supports the development and maintenance of distributed real-time systems.
- Extend the theoretical foundation for project management.
- Encourage the creation of associated project management tools to support incremental development of real-time systems.

8.4. Software Reuse

Several of the discussion groups identified issues and recommendations related to reuse. The recommendations in this area addressed issues such as the types of components that can be reused and disincentives to producing reusable components. The key recommendations, presented in previous sections, are reiterated below:

- Collect feedback on actual reuse experiences to identify the real problem areas.
- Develop techniques for identifying potentially reusable parts.
- Develop a framework for integrating components. This framework can then be evolved into a standard.
- Clarify the software data rights problems. Provide incentives to industry to move toward increased reuse.
- Perform research in the area of *reuse* with the goals of defining what types of objects can be reused (e.g., requirements specifications, certified flight code, validation test cases).
- Define techniques for packaging reusable components (e.g., libraries).
- Define schemes for acquiring knowledge of and locating reusable components.
- Prescribe methods and tools for designing and developing systems with eventual reuse in mind (e.g., domain analysis, specification and design methods).
- Define methods and tools for capturing the rationale associated with the design and implementation of system components.

8.5. Distributed Databases

Several of the discussion groups identified issues and recommendations related to distributed database technology. The recommendations in this area addressed issues such as the lack of effective consistency, concurrency, and synchronization mechanisms for distributed real-time databases and models for distributing data. The key recommendations, presented in previous sections, are reiterated below:

- Develop temporal databases.
- Develop object-oriented databases.
- Develop a theory of approximate query response.
- Develop techniques that allow the integration of semantic information into database systems.
- Orient the database research to address real-time performance issues as well as operational data integrity and fault tolerance.

8.6. Fault Tolerance

No workshop on parallel and distributed systems would be complete without some discussion of fault tolerance. Several of the discussion groups talked about fault tolerance, and their recommendations addressed issues such as the trend toward autonomous failure detection and correction in many new systems. The key recommendations, presented in previous sections, are reiterated below:

- Perform research in the area of *autonomous systems* with the goal of defining software and hardware architectures that permit increasing levels of functional fault tolerance through software and hardware fault tolerance.
- Define strategies and techniques for managing the use of redundant computing resources in the event of failures.
- Place an emphasis on establishing classes of faults where specialized knowledge can be employed to increase reliability and detect and circumvent faults (e.g., in the area of timing faults, scheduling theory may offer analytically based strategies for fault detection and correction).

8.7. Software Engineering Education

Although this topic was not heavily discussed, several of the discussion groups identified issues and recommendations related to software engineering education. The recommendations in this area addressed issues such as the need for domain-specific education and for graduating computer scientists with a well-rounded education. The key recommendations, presented in previous sections, are reiterated below:

- Develop a curriculum that educates computer engineers within the context of ultra-reliable underwater vehicles, so that they will understand that computational systems are "a means to an end, not just an end in themselves."
- Develop a curriculum that educates computer engineers in the skills of oral and written communication.
- Educate practitioners in basic software engineering methods.

8.8. Technology Transition

All of the discussion groups identified issues and recommendations related to technology transition. The recommendations in this area addressed issues such as the time lag between idea formulation and operational use of technology in systems, vehicles for better disseminating technology and information bi-directionally between industry and academia, and more effective use of the nation's research resources. The key recommendations, presented in previous sections, reiterated below:

- Develop a software analog to the very large-scale integration (VLSI) research process that moves the technology into standard practice in the 10-year time frame the way the very high-speed integrated circuits (VHSIC) program does. The VHSIC program produced a great deal of good research, but the transition suffered because of the lack of an application to demonstrate the technology. Therefore, an abstraction of an application is required to prov[illegible]e a framework in which to present the technology.
- Develop infrastructure mechanisms to share information locally and nationally. This could be accomplished with a national data repository that could be developed using existing infrastructures. The national data repository should be viewed as a national asset and probably be protected as such.
- Set up reward structures that encourage industry to "pull" in new technology and the academic community to "push" their results. An in-house individual should be responsible for pulling technology into the company.
- Encourage academic researchers to publish their results in applicable trade journals as well as professional journals.
- Reward academic researchers for multidimensional research and discourage unidimensional research.
- Establish working partnerships between industry and academia early enough to capitalize on new and current research:
    - Use the Software Engineering Institute (SEI) and the Aerospace Industries Association (AIA) as a catalyst for establishing partnerships.
    - Tie the partnership to a specific technical domain rather than to a specific system or project.
    - Involve multiple participants in the partnerships, not just a single industry-academic pair.
- Use the SEI to aid in identifying and disseminating appropriate technology. Ensure that there is a feedback loop between industry and academia.
- Develop a cooperative interface between government, industry, and academia, one that allows academia to receive input from industry and industry to more readily access and apply the results from academia.
- Define mechanisms for matching suppliers of research and technology with potential users of such research and technology at both the national and corporate levels.
- Define mechanisms to widely disseminate positive applications of advanced technologies as well as the lessons learned from attempting to do so.
- Define mechanisms for rapidly disseminating knowledge of technological and research advances.
- Identify and use, as a foundation, research that already exists. For instance, capitalize on the emerging theory of scheduling to derive subsystem-level design.
- Transition the rate monotonic scheduling theory, a well-established and growing body of theory that provides an analytical framework for managing time in real-time systems.

This theory is being developed at the Software Engineering Institute (SEI) and Carnegie Mellon University (CMU).

9. Conclusion

This AIA/SEI workshop was unique in its purpose and its execution. This was the first time the AIA and SEI have combined their resources to conduct a workshop, and the workshop succeeded in defining recommendations for the research community. It is anticipated that these recommendations will be given serious consideration by the government agencies that sponsor this type of research. Many of the participants in the workshop were representatives from such agencies.

In addition to providing normal distribution for this report, the SEI is providing an extended distribution that is intended to reach persons who support or conduct software research. It is hoped that the constructive recommendations resulting from this workshop will be of significant benefit to the effectiveness of the nation's software research efforts in helping to achieve successful software-critical systems in the 1990s.

Acknowledgements

The workshop from which the contents of this report were derived was made possible through efforts contributed by many persons. The authors gratefully acknowledge the major contributions made prior to and during the workshop by the following persons:

Workshop Planning Group Members

Clyde Chittister, Software Engineering Institute
Donald Paul, Aerojet Electro Systems
Larry Perkins, Martin Marietta Astronautics
Allan Whittaker, Honeywell Aerospace and Defense

Application Area Example Presenters

Carl Mohrman, Martin Marietta Air Traffic Control
Lawrence Klein, Aerojet Electro Systems
Eugene Opittek, Hughes Aircraft
John Shaw, Boeing Commercial Airplanes
Allan Whittaker, Honeywell Aerospace and Defense
Donald Wilson, Lockheed Missiles and Space

Discussion Group Leaders

Lawrence Perkins, Martin Marietta Astronautics
Eugene Opittek, Hughes Aircraft
John Stuelpnagel, Westinghouse Electric
Nelson Weiderman, Software Engineering Institute
Allan Whittaker, Honeywell Aerospace and Defense
Donald Wilson, Lockheed Missiles and Space

Appendix A: List of Participants

AIA/SEI Workshop
September 13-14, 1989

Bernard Abrams
Engineering Specialist
Grumman Corporation
Aircraft Systems
MS B35-35
Bethpage, NY 11731
(516) 575-8477

Mario Barbacci
Senior Member of Technical Staff
Software Engineering Institute
Software Systems
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7704
mrb@sei.cmu.edu

William J. Brownlow
Automation Systems Engineer
The Boeing Company
Boeing Aerospace & Electronics
P.O. Box 3999
M.S. 82-44
Seattle, WA 98124-2499
(206) 773-1050

Clyde Chittister
Director, Software Systems Program
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7781
cc@sei.cmu.edu

Vance Christiaanse
Systems Engineer
General Electric Company
Submarine Combat Systems
EP-7MD4
P.O. Box 4840
Syracuse, NY 13221
(315) 456-0765

Wesley W. Chu
Professor & Chair
University of California, Los Angeles
Computer Science Department
3731 Boelter Hall
Los Angeles, CA 90024-1596
(213) 825-8878
wwc@cs.ucla.edu

Robert P. Cook
Associate Professor
University of Virginia
Department of Computer Science
Thornton Hall
Charlottesville, VA 22903
(804) 982-2215
cook@cs.virginia.edu

Philip C. Daley
Manager, Adv. Computing Technology
Martin Marietta
Space Systems Company
P.O. Box 179
M/S 4372
Denver, CO 80201
(303) 977-4443

Lawrence Dimeo
Northrop Corporation

Larry E. Druffel
Director
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7740
druffel@sei.cmu.edu

Thomas Faught
Executive Director
National Center for Advanced Technology
1250 I Street
Washington, DC 20005
(202) 371-8455

Doug Ferguson
Supervisory Software Engineer
Westinghouse Electric Corporation
Electronic Systems Group
Development & Operations Division
P.O. Box 746, Ms 5370
Baltimore, MD 21203
(301) 765-5738
ferguson%eclus.dnt%tron.uucp
@umbc3.umbc.edu

Robert Firth
Senior Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-6305
firth@sei.cmu.edu

Charles Fisher III
Program Staff Specialist Sr.
Aerojet Electro Systems
Engineering
1100 W. Hollyvale Avenue
Azusa, CA 91702
(818) 812-1307

Mike Gagliardi
Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7738

Michael J. Gardner
Senior Engineer
Westinghouse Electric Corporation
Development & Operations Division
P.O. Box 746
MS 5370
Baltimore, MD 20707
(301) 993-8963

John R. (Jack) Garman
Associate Director (MSD) for I/S Planning
NASA
Johnson Space Center
Mission Support Directorate/FA
NASA Rd. 1
Houston, TX 77058
(713) 483-6231

Hassan Gomaa
Professor
George Mason University
Information Systems & Systems Engr.
4400 University
Fairfax, VA 22030-4444
(703) 323-3530

John B. Goodenough
Senior Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-6391
goodenough@sei.cmu.edu

Ronald Green
Electronics Engineer
United States Army
Strategic Defense Command/
GSTS Project
P.O. Box 1500
CSSD-H-GD
Huntsville, AL 35807-3801
(205) 722-1718

Robert L. Harris
Supervisory Electronics Engineer
United States Air Force
Avionics Laboratory
System Avionics Division
(WRDC/AAAF-3)
Wright-Patterson AFB, OH 45387-6543
(513) 255-3947

Walter L. Heimerdinger
Senior Research Fellow
Honeywell
Systems & Research Center
(MN65-2100)
3660 Technology Drive
Minneapolis, MN 55418-1006
(612) 782-7319
heimerdinger@svc.honeywell.com

David W. Hislop
Program Manager,
Computer Science & Engineering
United States Army
Research Office / Electronics Division
SLCRO-EL
P.O. Box 12211
Research Triangle Park, NC 27709
(919) 549-0641
hislop@brl.arpa

Dick Hotz
Chief Software Architect Joint Stars
Grumman Corporation
Grumman Melbourne Systems Division
2000 NASA Blvd. West
MS RC 2-220
Melbourne, FL 32904
(407) 951-6674

H. Stephen Hutchinson
Software Technologist
General Electric Company
Aerospace Electronic Systems
Strategic Systems Department
P.O. Box 1000
Blue Bell, PA 19422
(215) 354-3186

Lawrence A. Klein
Systems Manager
Aerojet Electro Systems
Environmental & Tactical Systems
P.O. Box 296
Azusa, CA 91702
(818) 812-2123

Mark H. Klein
Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7615
mk@sei.cmu.edu

John C. Knight
Associate Professor
University of Virginia
Department of Computer Science
Thornton Hall
Charlottesville, VA 22903
(804) 982-2216
knight@virginia.edu

Donald Krantz
Senior Principal Development Engineer
Honeywell
Ordnance
5901 Lincoln Drive
MN50-4900
Edina, MN 55436
(612) 939-2288

John P. Lehoczky
Prof. & Head, Dept. of Statistics
Carnegie Mellon University
Department of Statistics
Baker Hall 232G
Pittsburgh, PA 15213
(412) 268-8725
jpl@k.gp.cs.cmu.edu

Ted Lewis
Professor of Computer Science
Oregon State University
Computer Science Department
Computer Science Building
Corvallis, OR 97331-3902
(503) 737-3273
lewis@mist.cs.orst.edu

Larry C. Lindsey
Manager, Software Engineering
Aerojet Electro Systems
1100 West Hollyvale Street
P.O. Box 296
Azusa, CA 91702
(818) 812-1281

Reed Little
Member of the Technical Staff
Software Engineering Institute
User Interface Prototype
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-5792
little@sei.cmu.edu

Carl C. Mohrman
Senior Engineer
Martin Marietta
Air Traffic Control
475 School Street, SW
Mail Stop W7410
Washington, DC 20024
(202) 646-5746

Al K. Mok
Associate Professor
University of Texas at Austin
Computer Science Department
Tay 3.140C
Austin, TX 78712
(512) 471-9542
mok@cs.utexas.edu

Gilbert Myers
Head, Distributed Systems Branch
Naval Ocean Systems Center
Code 413
271 Catalina Blvd.
San Diego, CA 92152-5000
(619) 553-4136

William Novak
Resident Affiliate
Software Engineering Institute
Software Methods
5000 Forbes Avenue
Pittsburgh, PA 15213
wen@sei.cmu.edu

Eugene W. Opittek
Manager, Technical Laboratory
Hughes Aircraft Company
Command & Control Systems Division
1901 West Malvern Ave. P.O. Box 3310
Building 606, MS M236
Fullerton, CA 92634-3310
(714) 732-3101

Larry B. Perkins
Manager, New Business & Strategic Planning
Martin Marietta
Astronautics
P.O. Box 179, M/S L0330
Denver, CO 80201
(303) 971-6743

Marc Pitarys
United States Air Force
Avionics Laboratory
WRDC/AAAF-3
Wright-Patterson AFB, OH 45387-6543
(513) 255-3947

Walt Scacchi
Assistant Professor
University of Southern California
Computer Science Department
University Park
Los Angeles, CA 90089-0782
(213) 743-7424
scacchi@pollux.usc.edu

Lui Sha
Senior Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-5875
sha@sei.cmu.edu

Alan C. Shaw
Professor of Computer Science
and Engineering
University of Washington
Department of Computer Science
and Engineering, FR-35
Seattle, WA 98195
(206) 543-9298
shaw@cs.washington.edu

John L. Shaw
Supervisor Avionics/Flight Systems
The Boeing Company
Boeing Commercial Airplanes
New Airplane Division
P.O. Box 3707
Seattle, WA 98124-2207
(206) 234-8377

Charles Shelor
Staff Engineer
LTV Missiles & Electronics Group
Electronic Systems
P.O. Box 650003; MS WT-17
Dallas, TX 75265-0003
(214) 266-7791

Daniel P. Siewiorek
Professor
Carnegie Mellon University
SCS and ECE
Wean Hall
Pittsburgh, PA 15213-3890
(412) 268-2570
arpanet:dps@a.gp.cs.cmu.edu

David R. Smith
Sr. Engineer, Electronics
McDonnell Douglas
McDonnell Aircraft Company
Dept 312, Bldg 66, MC 0643203
P.O. Box 516
St. Louis, MO 63166-0516
(314) 777-2832

James G. Smith
Program Manager,
Applied Mathematics & Comp. Science
Office of Naval Research
Applied Research & Technology Directorate
Code 1211
800 N. Quincy Street
Arlington, VA 22217-5000
(202) 696-4715
jsmith@nswc-wo.arpa

Will Stackhouse
Assistant for High Leverage Technology
United States Air Force
Space Systems Division (AFSC)
c/o Jet Propulsion Laboratory (M/S 79-23)
4800 Oak Grove Drive
Pasadena, CA 91109-8099
(818) 354-1668

Jack Stankovic
Associate Professor
University of Massachusetts
Dept. of Computer & Information Science
Amherst, MA 01003
(413) 545-0720
stankovic@cs.umass.edu

Jay K. Strosnider
Asst. Professor - ECE
Carnegie Mellon University
ECE
Schenley Park
Pittsburgh, PA 15213
(412) 268-6927
jks@gauss.ece.cmu.edu

Henry G. Stuebing
Technical Consultant
Naval Air Development Center
Systems & Software Technology
Code 70C
Warminster, PA 18974-5000
(215) 441-2314
stuebing@nadc

John Stuelpnagel
Manager
Westinghouse Electric Corporation
Digital Systems
Box 746, MS 5230
Baltimore, MD 21203
(301) 765-6557

William L. Sweet
Manager, Industry Sector Operations
Software Engineering Institute
Technology Transition
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7706
ws@sei.cmu.edu

Sharilyn A. Thoreson
Section Chief - Avionics Technology
McDonnell Douglas
McDonnell Aircraft Company
P.O. Box 516
St. Louis, MO 63166
(314) 234-2089

Jane Van Fossen
Computer Technology Area Manager
Office of Naval Technology
Code 227
800 N. Quincy Street
Arlington, VA 22217-5000
(202) 696-4791
vanfosse@nrl-css.arpa

Roger Van Scoy
Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7620
rlvs@sei.cmu.edu

Robert Veltre
Member of the Technical Staff
Software Engineering Institute
REST Project
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-5883
rav@sei.cmu.edu

Nelson Weiderman
Member of the Technical Staff
Software Engineering Institute
REST Project
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7789
nhw@sei.cmu.edu

Charles B. Weinstock
Member of the Technical Staff
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7719
weinstock@sei.cmu.edu

Thomas Wheeler
US Army CECOM
Ft. Monmouth, NJ 07703-5000

Isaiah White
Specialist Engineer
The Boeing Company
Aerospace & Electronics
P.O. Box 3999
M/S 8H-09
Seattle, WA 98124
(206) 773-0572

G. Allan Whittaker
Director, Advanced Systems & Technology
Honeywell
Aerospace and Defense
Underseas Systems Division
600 Second Street NW
MN11-2043
Hopkins, MN 55343
(612) 931-3920

Donald A. Wilson
Director, Data Systems Engineering
Lockheed Missiles & Space Co., Inc.
Space Systems Division
1111 Lockheed Way
Sunnyvale, CA 94089-3504
(408) 743-1035

William Wood
Senior Software Engineer
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
(412) 268-7723
wgw@sei.cmu.edu

Grady C. Wright
Deputy General Manager
TRW
Command Support Division
One Federal Systems Park Drive
Fairfax, VA 22033
(703) 968-1022

R. M. Yanney
SEDD Director of Technology
TRW
Systems Engineering & Development Division
One Space Park
M/S DH2/2328
Redondo Beach, CA 90278
(213) 812-6033

Andre M. van Tilborg
Director, Computer Science Division
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217-5000
(202) 696-4312
avantil@nswc-wo.arpa